An Insight into the Philosophy of Computer Languages

Victor Otieno Mony, Franklin Wabwoba

Department of Information Technology, Kibabii University, Bungoma, Kenya

DOI: https://doi.org/10.51584/IJRIAS.2023.8805

Received: 03 August 2023; Revised: 11 August 2023; Accepted: 15 August 2023; Published: 10 September 2023

ABSTRACT

Computer programs don't exist in a vacuum. They belong to a rich-socio technological context which helps in gaining an in-depth understanding of the operations and functionalities of a computer. Computers have changed how societies think and behave. Developing countries, though lagging in technological advances when compared to other countries, have been transformed by technology. However, in such countries, knowledge of computer programming, its ontologies, and epistemologies is a reserve of the few privileged scholars. Perhaps, this is due to the limited access to technology caused by the economic crisis present in third-world countries. The work of philosophy is to present knowledge in an acceptable way that will enhance the advancement and recreation of ideas thus, a philosophical perspective on programming languages for developing countries is a step in the right direction as it will bud advancements in computer language research and arouse interest in the philosophy of computer languages. This paper presents an atmosphere where a broader understanding of the philosophical concepts in computer programing languages can be derived. It gives an overview of the foundation of programming languages, highlights their developments over time, and gives the philosophical stand of the research works that have delved into the field. This paper also examines methods through which other academic disciplines have enabled the developments and advancements in computer programing languages and the contribution of computer programing to other disciplines. Should a reader gain an unbiased perspective on the debate surrounding the philosophy of computer programming languages, then, this paper would have served its purpose.

Keywords: Philosophy of Programming Languages, Ontology of Computer Programing Languages, Epistemology of computer programing languages, Object Oriented Programming, Visual Programing, Programing Language Syntax.

INTRODUCTION

Throughout history, man has always been involved in the process of searching for ways of communicating with his objective and subjective realities. This has been largely achieved through language [1]. Language is a critical component of the lifestyle of man. Without language, there is no understanding [2]. In the field of research, when delving into the philosophy of language, there is much that needs to be taken into account. For instance, language stands as the central means of communication. Language entails phenomena such as speech correlation, thinking, symbolism, and significance among others [1].

Language matters even if a person on the street may think it doesn't [2]. It is for this reason that many academic fields have included it in their study. A broad definition of language is a developing semiotic system. Language is either the natural human language used for man-to-man communication and knowledge transmission or artificial language that is used in computer programming [1].

Ingrained into the philosophy of Information Technology, is the analogy of programming languages. Computer Programming languages, are also known as high-level languages and were created to create abstractions that can be comprehended by humans to enable an efficient translation point between humans and computers [3].

A problem faced when studying programming languages, especially in developing countries is the role of mathematics and formal logic in computer programming. Is there a relationship between the two? And if there is, how do their artifacts and formal models relate? [4], [5]. As Turner adds to the questions, is a programming language purely mathematical? Turner feels that in practice, there are strong design aspects that do not fit into mathematical philosophies and that should programming languages be grouped with mathematical philosophies, accounting for the role of machines would be impossible [6].

So critical are programming languages that every person in the field of computing should understand and appreciate their functions. These languages are not only a means of passing instructions to the computer but also a means through which thinking and communication take place. Furthermore, programming languages have continually shaped the field of computation [3]. An understanding of the relationship between computer programming languages and other academic disciplines is equally important in ascertaining their relevance in philosophical studies.

From the beginning, classical programming languages were viewed as complex and far in approximation from the natural language [7]. The first programming language, known as Simula 1, was a language that enhanced the representation of real-world systems through simulations and was viewed as a tool for both programming and communication [3]. Due to the complexity of programming languages, it has never been easy to improve the programming process [7]. Research work developed from Simula 1, and advanced over the years to make computer languages easy to read, print, and more useful for communication [3].

Computer programing languages present a lot of philosophical questions and challenges. For instance, what is the nature of the physical components concerning objects in programming languages? How does the linguistic program determine matter? What can be considered the ontology and the epistemology of computer programming? [6]. Knowledge-based programming is in its infancy and this makes programming languages have great limitations to natural language. Understanding the philosophy of programming languages is ideal for teaching programming and its development toward natural language syntax [7].

Unfortunately, as it stands currently in classical programming, a man without a programmer’s proficiency can hardly manipulate software functionality [1]. Therefore, one of the central interests of the philosophy of information technology in developing nations remains how the communication between man and machine can be made intuitive.

The need to find ways through which non-programmers can communicate with the information environment is eminent [1]. It is also of concern to understand the compositional nature of semantics which though expressed in English, can be developed more formally through operations of underlying abstract machines [6]. The difference between programming semantics, their interpretation, and implementation needs to be made visible to easily ascertain what is correct and what is false. This will make it easy to identify failures and enhance faster development in programming languages machine [6].

METHODS

This research study has its foundation laid on refereed research literature. The research starts by an extensive search for relevant literature. For the purposes of literature search and published papers identification, this study utilized the Preferred Reporting Items for Systematic Reviews and Meta-analyses (PRISMA) because it offers transparency in identification, selection and synthesis of different studies.

Literature search was conducted through a search of peer-reviewed journal literature and conference papers in five databases namely IEE Explore, Google Scholar, ACM Digital Library, Science Direct and Web Science. An application of the search string keywords “Computer Programming Languages

Philosophy", "Ontology of Computer Languages", and "Epistemology of Computer Languages" on the said databases revealed that research work has not covered much on the philosophy of programming languages.

The search results returned 78 papers, 50 of which were in the grey areas of the intended research paradigm. The total number of research papers deemed directly relevant to the area of study were 28. A quick, and comprehensive study for relevance and for deduction of a publication period less than ten years, was performed by the researchers on the papers and this further knocked out 6 papers leaving a total of 22 papers. Further critical examination of remaining papers was done for relevance, correctness and importance of information to the research problem. This examination left the researchers with 12 papers that have been used effectively to reference the work of this study. The 12 research papers used in this study range from the years 2016 to 2021.

The views of different researchers from across the papers were considered, conceptualized, discussed and a proper conclusion drawn based on the ensuing discussions.

RESULTS

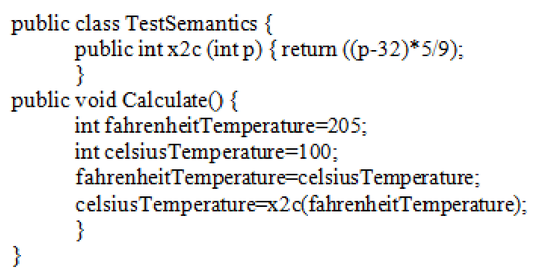
There is a huge difference between man-to-man communication and man-to-machine communication. Man-to-man communication happens through natural language while the man to machine language is artificially done through a created computer programming language [1]. Computer programming languages, despite having different development processes from written and spoken languages, remain a creative and evolving system of communication between man and machines. An effective programming language has to be interpreted and compiled into a binary code through a given set of rules. The idea behind the creation of high-level programming languages is to ease the interaction between humans and machines by ensuring effective communication and implementation of human ideas on machines [3].

When it comes to man-to-machine communication, instructions are transferred by man through his language and interfaces. However, the machine accepts the language and translates it into binary to understand and enact it [1]. Despite their complexity, almost all programming languages have a method through which lines are marked and commented on for an easy explanation of the purpose and function of the program code to others. The challenge with programming languages has always been how to divide complicated programs into smaller subsets to ensure that other programmers can make sense of the whole language [3].

Semantics

Programming language semantics has its roots in mathematical logic. Some philosophers believe that semantics inherited its technical background from model theory which has its language philosophy originating from the works of Frege [6]. Semantics refer to the processes followed by a computer to execute a given program code of a programming language [7]. A good example of an algorithm which adheres to JAVA program semantics is as indicated in figure 1:

Java Program Semantics Example



```
public class TestSemantics {
        public int x2c (int p) { return ((p-32)*5/9);
        }
public void Calculate() {
        int fahrenheitTemperature=205;
        int celsiusTemperature=100;
        fahrenheitTemperature=celsiusTemperature;
        celsiusTemperature=x2c(fahrenheitTemperature);
        }
}
```

Figure 1: Java Program Semantics Example

Semantics is represented as a set of concepts involving actions, resources, and relations between machinereadable language and program codes.

Semantics involves how these are presented, handled, and retrieved by compilers through given sets of actions [7]. By their compositions, semantic values in complex expressions are fixed through their linguistic structure making them a central part of Frege's theory of meaning [6].

In the semantics of programming languages, actions are connected with elements that respond to questions such as who, where, when, what, how, from what, by what, how many, for what, and to whom. Brackets are used to group symbols of a similar association. Subjects are associated with other elements through questions such as from what, for what, and whose while semantic Relations are created when two or more entities are to be conjoined together [7].

Some philosophers hold to the idea that semantics originated from Wittgenstein's philosophy of language. Wittgenstein's philosophy argues normatively that semantic theory has to distinguish between correct and incorrect use of language through given decisions [5], [6]. Frege's and Wittgenstein's points of view form a serious basis for any philosophical arguments in semantic theory.

There also exists the relativist account of semantics which is liberal. Semantic relativism states that anything can be in a semantic domain depending on how it is handled within a programing language. This can relate well with normative semantics if specification on the consequence of choices is considered [6]. Thus, a philosopher can argue that there exists a difference between programming semantics and its implementation.

- **The Ontology of Programing Languages**

Ontology is the study of being and its assumptions constitute what is considered as reality. One of the major questions in ancient Greek philosophy was what the nature of being is. Plato's doctrine which evolved around the theory of ideas appeared to some extent to have answered this question. Heraclitus however stated that the nature of being lies in change even as Parmenides reiterated that the real world doesn't change [8]. Even in developing countries where there has been a slow uptake of technology due to cyber security issues [9], and limited knowledge in programming, Parmenides has been proved wrong by technological developments. Indeed the only constant is change.

In computer programming languages, ontology is the semantic description of domain concepts that provides a formal specification of terms in a given domain and the relations existing between the said terms [10]. Programing language ontologies solve the problem of resource selection and allocation in a computer program. How for instance do you choose a subset from a set of subjects? Do you choose randomly, successively, through ranking, or by application of a given set of rules and restrictions? Understanding language ontology is key to resolving this issue during programming [7].

Researchers have proposed numerous ontological perspectives for the programming domain. Generally, all the proposed ontologies focus on enhancing the understanding and querying of programming languages. Most ontologies in this domain focus on language element representations, especially in object-oriented programming languages and most researchers have focused on Java and C# Programming languages [10].

Ontologies are useful in the management of software and learning programming languages with their underlying technologies. In Computer programming, ontologies represent general software development concepts such as control structures, standard libraries, development platforms, and data formats [7].

The reason why program codes are mapped into an ontology is to provide a simplified understanding of the said programs. In an Object Oriented Programming language such as Java, graphs can be used to describe the code where identifiers such as names of variables, functions, and classes are the vertices of the graph.

The relationships between identifiers are the edges of the whole graph. Homomorphism between the graphs can also be used to represent ontology [7].

Turner views the ontology of programs as technical artifacts with a dual nature. Programming languages have functional and structural descriptions. Functional descriptions are the specifications of the artifacts while structural descriptions inform us of the physical properties of the programs. Functional components are identified with functional descriptions while structural components refer to the programming language enriched with the interpretation of its semantics. This means that programming languages have both abstract natures separate from their physical nature. This forms the backbone of the ontology of programming languages [6].

Variable names mean nothing to machines and are not even necessary in the understanding of program code. However, they have fundamental cultural meanings [4]. From our personal experience, in developing countries, variable names are attached to the social-cultural environment of the programmer.

- **The Epistemology of Programming Languages**

Epistemology is the nature and forms of knowledge and deals with how knowledge is acquired, and disseminated in a given field of study. The epistemology of programming languages borrows heavily from the world of classes and objects, their characteristics, and their inheritance. There are many occasions where the field of computer science has borrowed concepts from other disciplines. One such occasion is the Artificial Neural Network which has been inspired by the explanations of neurons in biology [8].

What therefore is the epistemological status of computer languages? This involves the study of the relationship between symbols in computer programs, their specifications, and their physical manifestations. Within the epistemology of programming languages is the idea of correctness which deals with an analysis of whether the program symbols and physical implementation satisfy their sets of given specifications [6]. The following are considered when discussing the epistemology of programming languages:

- Correctness: Epistemology involves some kind of reasoning. Philosophers have thus tried to consider reasoning within programming languages correctness as part of scientific reasoning. What is the consideration to say that a programing language is correct?

There are standards of measure for correctness. First, it must meet the mathematical nature of proofs, second, it must meet the mechanical challenge which is the use of computers to ascertain the mathematical proofs. Thus, there is the pragmatic challenge that looks into the practicality of the verifiability of code symbols. Finally, the scientific challenge states that there has to be experimental knowledge that enables testing to ascertain computational artifacts [6].

In modern software development, proof of correctness must rely on physical computational process analysis. Turner calls it the mechanical challenge. The formal relations between needed program specifications and the symbolic demands should also be done to arrive at correctness. There exists a mathematical approach to correctness which is affected by the complexity of a programing language [6].

- The Object-Oriented Paradigm: Object Oriented programing makes use of ‘objects’ to simplify the programing process. A good example of an OOP is the Java Programing language. In the philosophy of programming languages, objects are a means of thinking and writing programs. Notable is the fact that early computer programs were written in machine codes that operated on memory locations and performed only processor-supported operations such as add, multiply, and Boolean [3].

Early programming languages were procedural languages where program constructs were sequential sets of commands executed one after the other. The introduction of structured programming enabled programmers to re-use parts of given programs and enable better coherence in programming. In the 1960s, the Simula programing language, which was developed by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Center, was the first of a kind to start the incorporation of objects in programming languages [3], [8], [11]. The Simula programming language is the source of several modern-day programming languages like C, Java, C#, Fortran, Python, PHP, Ruby, Pearl, and Delphi [8].

Nygaard and Dahl developed an initial compiler framework that underwent significant revisions before it could become functional. In the original concept of language, there was a series of stations handling a given queue of customers. Customers were passive data structures containing only variables which interacted with active station processes [3].

Object Oriented Programming (OOP) language is one of the most successful developments in programming history. Implementation of OOP languages varies by a given language though generally, all OOP programming languages make use of classes and objects. Classes exist throughout the lifetime of a program and cannot be altered during runtime. On the other hand, objects are freely created, altered, and destroyed at runtime. An object’s capability and structure are determined by a class these classes are the blueprints for creating objects. Classes form a hierarchy through inheritance through which other classes are derived. A single class can have many objects created within it [8], [11].

OOP programming languages refer to the style of writing computer codes and the unique manner through which digital objects are created and presented in programming languages. OOP has shifted programming languages from being a descriptive language of individual bits, inputs, and outputs to a language of abstract concepts such as functions, classes, graphics, and objects. OOP languages have complex data types such as Strings, arrays, lists, and complex programmer-defined data structures [3].

Object-oriented programing has significant philosophical implications in computer languages. To appreciate the philosophical impact of OOP, an in-depth understanding of the relationship between objects and language is necessary [3]. The philosophy of OOP is grounded in the following facts; there exist classes and objects, classes are eternal, objects can be created and destroyed, objects depend on classes which are models for objects, the classes form a hierarchy and one class corresponds to many objects, and one class exists at the top of the hierarchy [3], [8].

The idea around OOP is to provide an abstract means of thinking about the real work of a program [3]. However, nominalists’ view is that universals do not exist in the real world and are instead viewed as abstracted from real objects by apprehending similarities between them. To nominalists, the only existence, in reality, is that of physical objects and not their properties [8].

Demey states that Tylman’s analogy between Plato’s universalism and the OOP paradigm seems reasonable. In OOP, a class can have many subclasses inheriting from it but cannot inherit from multiple super-classes. The ideology of multiple inheritances is an extremely controversial topic in object-oriented programming and brings about complex problems which are difficult to solve [11]. Denying that programming languages are real, and creative, and narrating a story about the world just like other languages is not possible.

In the field of programming, objects were invented to abstract the real world. In the world of computer languages, what is real is not physical but rather virtual objects (software) controlling the physical (hardware). The language of a computer consists of numbers comprised of bits which are merely a difference in voltage and light intensities underneath [3].

Some philosophers insist on the ‘concrete existence’ of objects in programming languages. Object-oriented ontologists propose that programming languages are analogous to human language and thus go beyond the human understanding of philosophy [3]. From my analysis, Objects in programming languages are unique, and a programing language is just like any other language in which these unique objects exist.

DISCUSSIONS

Plato introduced into philosophy the concept of ‘a problem of universals’ which stated that there exists separate entities influencing the observable world. According to Universalism, the redness of an apple is because of the existence of ‘redness’ as a universal abstract. This concept, though not universally accepted, has remained a subject of discussion to date. The main debate question with this is the nature and existence of universal entities [8]. In computer language philosophy especially to OOP, a program is a set of objects consisting of properties and functions that enable their interactions. This not only fits into the concept of universalism but also forms the central philosophy of object-oriented programming [3]. In programming, the universals proposed by Plato and Aristotle are the common properties of objects. Medieval philosophy took a realistic stand and accepted that universals exist and are separate entities [8].

The ability to compile a programming language together with its sets of translation rules into binary code is a great pediment to human-computer communication. However, through object-oriented programming, the complexity between human and computer languages is greatly reduced [3]. There is still much more work to be done and to enhance the development of effective programming, especially in developing countries, it is ideal that a programming language that translates the real world into computer language must be created.

Most languages incorporating OOP utilize class dualism. However, some languages take the prototype-based programming approach where there are no classes and objects created to have direct specifications of properties and behavior. Just like in the theory of ideas, nominalism is also found in programming languages. In OOP languages, the programmer can apprehend similarities between existing objects and exploit their common properties and behavior [8].

When programming languages are placed under a critical analysis, there are those philosophers who try to separate language from code. Such philosophers argue that computer languages are merely sets of instructions since they are compiled into machine code through sets of strict rules. To them, this means that programming languages cannot signify anything just like real human languages [3]. In Turner (2020), Rapaport argues that programs are not technical artifacts because programming languages do not need to have functional descriptions. However, turner suggests that this argument does not prevent a person from undertaking the study of language syntax and mathematical objects in their own right in programming languages [6]. In our opinion, it is good to note that programming languages have their semantic interpretation and structural description of their different machine interpreters. Turner adds that computer science does not have technical artifacts since it consists of abstract immaterial things, a situation that Turner calls the theoretical perspective on the nature of computer science [6].

- **The Philosophy of Programming Language Design**

Programming language design has its given philosophy which examines the specific aims and problems concerning fundamental philosophical questions.

What is the nature of reality in a programing language design? How do we judge a design? Turner states that simplicity, modularity, and abstraction are the three general principles guiding the programming language design process [6].

According to Turner, whereas logic programing is a legitimate discipline in computer science, there is a lack of clarity as to whether it forms part of philosophy. Logic programing is driven by immaterialism but has practical goals aiming to change the nature of programming. However, logic programing is part and parcel of the programing language process. Turner concludes that it is a theory and not a philosophy of programming languages [6].

- **The Visual Programming Language Design**

One of the ways through which programing language has been made accessible for non-professional programmers in developing countries is through the inclusion of attributes that are generally accepted in human society. The application of visual symbols to programming has increased the acceptance and utilization of programming in modern society. Non-professionals utilize the visual computer language as they program animation management algorithms [1].

Object-oriented programming was the major advancement that transformed computers from calculating machines. Visual programming introduces the idea of multiple subjectivities into programming languages and points out ways through which these subjectivities are created. This is of critical importance to object theory [3].

The known traditional textual programming languages are moving towards visual interfaces. Object-oriented programing where programs have properties and characteristics describing object behavior has taken the stage. This has promoted the development of visual tools into software development. The visual tools have minimized the quantity of written textual program code and have helped in setting object properties in programming languages [1].

Digital objects have both virtual and visual histories. Objects in programming languages arise out of a desire to create a model of the world within the computer and out of a desire to create a visual world. OOP gives programmers the power to write functions and data structures for objects and let them function on external aspects of the world [3]. This makes programming languages an important source of technological philosophical insights.

The use of language sometimes involves a complex relationship with computations. In the digital world, there are fundamentally flawed algorithms that are a typical part of writing. This is known as misidentification. A good example of misidentification is in the construction of easy-to-guess passwords which make machines vulnerable to electronic theft. Language sometimes is seen as a deficient symbolic conduit that is sufficiently enriched in the world of computers and corrected through efficient algorithmic processing [12].

Harman was the founder of object-oriented ontology. His view was that the world is made up of objects through which philosophy speculates, especially regarding the interactions and implications of objects. Heidegger supports Harman's view by stating that objects have depths and are withdrawn from each other and thus can't be accessed in their totality. According to the two researchers, this is enough reason to state that objects belong to a part of philosophy known as speculative realism [3].

CONCLUSIONS

In developing countries, programing language philosophy plays an important role in the learning, teaching, development, and application of computer programing. It has helped in the maturity of software development across many domains. The ontology of programming languages can be organized into semantic models, actions, resources, and restrictions affecting computer programing. This makes it easy to structure algorithms in a manner that it is philosophically possible to understand and interpret.

Object-oriented programing has taken center stage in the philosophy of programming languages. An understanding of OOP leads to an understanding of the visual aspects of programming languages which in turn, helps in the concrete development of programing language philosophies and the ideas surrounding their usage.

Also good to note is that from an analysis of research literature, it can be rightly concluded that few researchers have discussed the philosophy of computer programing languages. Perhaps, because as per research analysis, it's a relatively new field of study.

REFERENCES

1. A. V. Moiseenko, I. V. Brylina, A. A. Kornienko, O. G. Berestneva, and N. N. Kabanova, "Visual language as a mean of communication in the field of information technology," IISA 2015 – 6th Int. Conf. Information, Intell. Syst. Appl., pp. 12–15, 2016, doi: 10.1109/IISA.2015.7388015.
2. H. Bones, S. Ford, R. Hendery, K. Richards, and T. Swist, "In the Frame: the Language of AI," Philos. Technol., vol. 34, pp. 23–44, 2021, doi: 10.1007/s13347-020-00422-7.
3. J. Joque, "The Invention of the Object: Object Orientation and the Philosophical Development of Programming Languages," Philos. Technol., vol. 29, no. 4, pp. 335–356, 2016, doi: 10.1007/s13347-016-0223-5.
4. T. Petricek, "Computing and Programming in Context—Introduction," Philos. Technol., vol. 34, no. 1, pp. 7–11, 2021, doi: 10.1007/s13347-020-00411-w.
5. G. Dennis, Karugu, R. Salome, Christine, and F. Wabwoba, "TOWARDS PHILOSOPHY OF INFORMATION TECHNOLOGY," Int. J. Contemp. Appl. Res., vol. 6, no. 6, pp. 1458–1467, 2019, doi: 10.4018/978-1-5225-8060-7.ch068.
6. R. Turner, "Computational Artifacts: the Things of Computer Science," Philos. Technol., vol. 33, no. 2, pp. 357–367, 2020, doi: 10.1007/s13347-019-00369-4.
7. V. N. Kuchuganov, D. R. Kasimov, and M. V. Kuchuganov, "Principles of organizing the semantic ontology of programming," RPC 2017 – Proc. 2nd Russ. Conf. Comput. Technol. Appl., vol. 2017-Decem, pp. 122–126, 2017, doi: 10.1109/RPC.2017.8168082.
8. W. Tylman, "Computer Science and Philosophy: Did Plato Foresee Object-Oriented Programming?," Found. Sci., vol. 23, no. 1, pp. 159–172, 2018, doi: 10.1007/s10699-016-9506-7.
9. N. A. Wechuli, F. Wabwoba, and W. Jotham, "Cyber Security Challenges to Mobile Banking in SACCOs in Kenya," Int. J. Comput., vol. 27, no. 1, pp. 133–140, 2017, [Online]. Available: http://erepository.kibu.ac.ke/handle/123456789/313%0Ahttp://erepository.kibu.ac.ke/bitstream/handle/123456789/313/Nambiro_Cyber security challenges to mobile banking in SACCOs in Kenya.pdf?sequence=1&isAllowed=y.
10. B. Diatta, A. Basse, and S. Ouya, "PasOnto : Ontology for Learning Pascal Programming Language," IEE Glob. Eng. Educ. Conf., pp. 749–754, 2019.
11. L. Demey, "The Porphyrian Tree and Multiple Inheritance: A Rejoinder to Tylman on Computer Science and Philosophy," Found. Sci., vol. 23, no. 1, pp. 173–180, 2018, doi: 10.1007/s10699-017-9531-1.
12. A. Klobucar, "Programming's Turn: Computation and Poetics," Humanities, vol. 6, no. 2, p. 27, 2017, doi: 10.3390/h6020027.